(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-13227
(P2004-13227A)
(43)公開日　平成16年1月15日(2004.1.15)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 17/50	G06F 17/50	664A	5B046
G06F 9/455	G06F 17/50	664B	
	G06F 9/44	310D	

審査請求 未請求　請求項の数 22　OL　（全 17 頁）

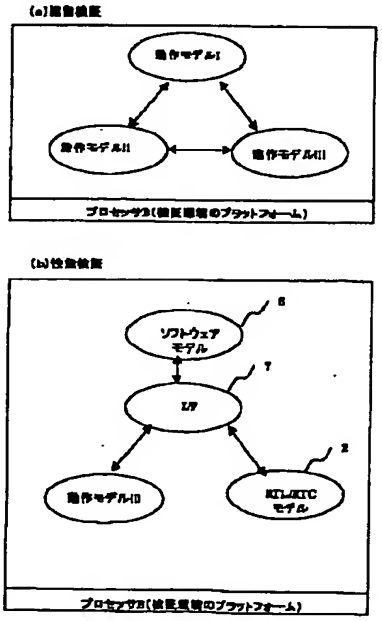
(21)出願番号　特願2002-162048(P2002-162048)
(22)出願日　平成14年6月3日(2002.6.3)

(71)出願人　000005821
松下電器産業株式会社
大阪府門真市大字門真１００６番地
(74)代理人　110000040
特許業務法人池内・佐藤アンドパートナーズ
(72)発明者　篠原　克哉
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72)発明者　水野　雅信
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72)発明者　本原　章
大阪府門真市大字門真１００６番地　松下電器産業株式会社内

ドターム(参考)　5B046 AA08　BA03　JA05

(54)【発明の名称】シミュレーション装置並びにシミュレーションモデル生成プログラム

(57)【要約】
【課題】キャッシュ解析の可能な抽象度を保った性能検証を、ＩＳＳ（命令セットシミュレータ）を必要とせずに、単一の検証プラットフォームにより高速に行うことが可能なシミュレーション装置を提供する。
【解決手段】ソフトウェア実装される機能を、検証環境プラットフォーム（プロセッサＢ）上で動作するソフトウェアモデル６として生成し、ハードウェア実装される機能を、ＲＴＬ／ＲＴＣモデル２またはこれより抽象度の高い動作モデルとして、プロセッサＢ上で性能検証を行う。各モデル間には、モデル間の入出力データを受け渡しするインタフェースモデル７を設ける。
【選択図】　図１



【特許請求の範囲】
【請求項1】
検証処理を実行する検証実行プロセッサを備え、検証対象システムが有する検証対象プロセッサの機能をシミュレートするプロセッサモデルと、前記検証対象システムにおける前記検証対象プロセッサの周辺ハードウェアの機能をシミュレートするハードウェアモデルとを、前記検証実行プロセッサ上で動作させることにより、前記検証対象システムをシミュレートするシミュレーション装置であって、
前記プロセッサモデルは、前記検証対象プロセッサの機能が、前記検証実行プロセッサで実行可能なコードで記述されたソフトウェアモデルであり、
前記プロセッサモデルと前記ハードウェアモデルとの間で入出力データの受け渡しを行うインタフェースモデルを用いることを特徴とするシミュレーション装置。
【請求項2】
前記インタフェースモデルが、モデル間でプロトコル変換を行うプロトコル変換部を備えた、請求項１記載のシミュレーション装置。
【請求項3】
前記インタフェースモデルが、前記プロセッサモデルとの間で、前記検証対象プロセッサ用の命令レベルまたはクロックレベルで入出力データを受け渡しする、請求項２記載のシミュレーション装置。
【請求項4】
前記インタフェースモデルが、前記プロセッサモデルとの間で、前記検証対象プロセッサ上で動作するプログラムの基本ブロックレベルで入出力データを受け渡しする、請求項２記載のシミュレーション装置。
【請求項5】
前記インタフェースモデルが、前記プロセッサモデルとの間で、前記検証対象プロセッサ上で動作するプログラムの関数レベルで入出力データを受け渡しする、請求項２記載のシミュレーション装置。
【請求項6】
前記インタフェースモデルが、前記プロセッサモデルとの間で、前記検証対象プロセッサ上で動作するプログラムのタスクレベルで入出力データを受け渡しする、請求項２記載のシミュレーション装置。
【請求項7】
前記プロトコル変換部が、前記モデル間のデータ転送の方向またはデータの転送単位の大きさを変換する、請求項２記載のシミュレーション装置。
【請求項8】
前記プロトコル変換部が、前記モデル間の入出力データを記憶する記憶部と、前記記憶部に対する入出力データの書き込みおよび読み出しを制御する制御部とを備えた、請求項２記載のシミュレーション装置。
【請求項9】
前記記憶部が、前記モデルからの入力データを記憶する部分と前記モデルへの出力データを記憶する部分とを独立して有する、請求項８記載のシミュレーション装置。
【請求項10】
前記記憶部がＲＡＭを用いて構成される、請求項９記載のシミュレーション装置。
【請求項11】
前記記憶部がＦＩＦＯメモリを用いて構成される、請求項９記載のシミュレーション装置。
【請求項12】
前記ハードウェアモデルが、前記周辺ハードウェアの機能をＲＴレベルで記述したＲＴＬ／ＲＴＣモデル、または、前記周辺ハードウェアの機能をＲＴレベルよりも高い抽象度で記述した動作モデルである、請求項８記載のシミュレーション装置。
【請求項13】

前記制御部が、前記プロセッサモデルと、動作モデルの遅延を、前記プロセッサモデルと前記ＲＴＬ／ＲＴＣモデルのプロトコルに合わせるために、前記プロセッサモデルと動作モデルの出力データの前記記憶手段における遅延を調整するものである、請求項１２記載のシミュレーション装置。

【請求項14】

前記ソフトウェアモデルを前記検証実行プロセッサで動作させた結果から生成されるプロファイル情報を、前記検証対象プロセッサのプロファイル情報に変換する、請求項１２記載のシミュレーション装置。

【請求項15】

請求項１に記載のシミュレーション装置で用いられるソフトウェアモデルを生成するシミュレーションモデル生成プログラムであって、

前記検証対象プロセッサの機能を記述したソースプログラムを、前記検証実行プロセッサで実行可能な命令セットを用いてコンパイルすることを特徴とするシミュレーションモデル生成プログラム。

【請求項16】

前記コンパイルの際に、前記ソフトウェアモデルから外部モデルへのトランザクションをクロックサイクルレベルで生成する、請求項１５記載のシミュレーションモデル生成プログラム。

【請求項17】

前記コンパイルの際に、前記ソフトウェアモデルから外部モデルへのトランザクションを命令レベルで生成する、請求項１５記載のシミュレーションモデル生成プログラム。

【請求項18】

前記コンパイルの際に、前記ソフトウェアモデルから外部モデルへのトランザクションを基本ブロックレベルで生成する、請求項１５記載のシミュレーションモデル生成プログラム。

【請求項19】

前記コンパイルの際に、前記ソフトウェアモデルから外部モデルへのトランザクションを関数レベルで生成する、請求項１５記載のシミュレーションモデル生成プログラム。

【請求項20】

前記コンパイルの際に、前記ソフトウェアモデルから外部モデルへのトランザクションをタスクレベルで生成する、請求項１５記載のシミュレーションモデル生成プログラム。

【請求項21】

前記コンパイルの際に、生成コードと前記ソースプログラムとの参照関係を抽出する、請求項１５記載のシミュレーションモデル生成プログラム。

【請求項22】

前記コンパイルの際に、前記ソフトウェアモデルを検証実行プロセッサで動作させた結果から生成されるプロファイル情報を保持するコードを挿入する、請求項１５記載のシミュレーションモデル生成プログラム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、大規模集積回路（ＬＳＩ）設計、特にシステムオンチップ化に対応するための設計に関するものであり、設計上流におけるシステム／アーキテクチャの性能評価を行うシミュレーション装置に関するものである。

【０００２】

【従来の技術】

従来から、電子機器は、例えばプロセッサ、メモリ、バス等の種類ごとのＬＳＩを、半導体チップ上に個別に形成した後、各チップをプリント配線基板などの母基板上に実装することによって、製造されてきた。

【０００３】

ところが近年、これに用いられる集積回路装置に対し、１チップ化、小型化、軽量化、省電力化および低コスト化の要求が高まっている。このような傾向は、特にデジタル情報家電分野において、より顕著にみられる。そして、これに応じて、半導体メーカーはその研究開発の重心をメモリからシステムＬＳＩへと移行してきている。

【０００４】

かかるシステムＬＳＩは、具体的には、メモリや各種の論理回路を単一のチップ上に設けるいわゆるシステムオンチップ化によって実現されている。そこで、このシステムオンチップ化に対応した設計技術として、ある機能を実現するブロック（いわゆる機能ブロック）を設計するためのデータを予め準備しておき、これらのデータを利用して、各機能ブロックを組み合わせた所望のシステムＬＳＩを設計する、という手法が提案されている。このような設計手法を用いた場合には、各機能ブロックについてはすでにその機能を実現するための構成が定められているので、集積回路装置設計の際には、各機能ブロック間の配線や周辺回路の設計を行うだけですむ。

【０００５】

システムの機能ブロックは、設計上流では動作レベル（アルゴリズムレベル）で記述された動作モデルによって記述され、この段階でシステムの機能検証が行われる。ついで、機能ブロックのそれぞれをハードウェアおよびソフトウェアのいずれで実現するかを決定し、ハードウェアで実現される機能ブロックの動作モデルをＲＴレベルで設計し、性能検証する。また、ソフトウェアで実現される機能ブロックは、そのソフトウェアを動作させるプロセッサのＩＳＳ（命令セットシミュレータ）上で動作させて性能検証を行う。

【０００６】

システムの性能検証のフェーズにおける従来のシミュレーションシステムを、図１７（ｂ）に示す。ここでは、動作モデルＩ～ＩＩＩによってそれぞれ記述された機能ブロックのうち、例えば、動作モデルＩをソフトウェアで実現し、動作モデルＩＩＩをハードウェアで実現することが決定したものとする。ソフトウェアで実現される動作モデルＩは、例えばＣ言語またはアセンブリ言語などで記述され、オブジェクトコードにコンパイルされることにより、プログラム１として生成される。生成されたプログラム１は、プロセッサＡのＩＳＳ５上で動作する。動作モデルＩＩＩは、動作モデルよりも抽象度の低いＲＴＬ（Ｒｅｇｉｓｔｅｒ　Ｔｒａｎｓｆｅｒ　Ｌｅｖｅｌ）で設計され、ＲＴＬ／ＲＴＣモデル２となる。この場合、システム中に、未だ動作モデルである機能ブロック３（動作モデルＩＩ）が存在するため、ＲＴＬ／ＲＴＣモデル２との検証を行うためのインタフェース４を用いて、システム全体の検証を行う。このような設計手法により、設計効率の大幅な向上を図れる。

【０００７】

【発明が解決しようとする課題】

しかしながら、上記した従来の設計手法では、次のような問題がある。

【０００８】

従来の、ソフトウェアで実現される動作モデルをＩＳＳ上で実行する方法では、ＩＳＳが存在することが前提条件であり、ＩＳＳの実行時間が性能検証時間の大半を占める。すなわち、図１７（ｂ）に示すように、プログラム１は、プロセッサＢ上で、プロセッサＡのＩＳＳ５が命令を解釈および実行するという煩雑な処理を行うことにより、シミュレーションされる。これにより、プログラム１の実行時間が、システムの性能検証の多大な時間を占めることになる。

【０００９】

一方、従来は、ＩＳＳを用いない、動作モデルのみによる性能検証では、キャッシュメモリの解析を行うことが可能な抽象度での検証が困難である。また、ＲＴＬレベルでの検証は、動作モデルでの検証に比べ、時間がかかる。

【００１０】

本発明は、前記従来の問題を解決するため、検証対象システムのプロセッサのキャッシュメモリの解析を行うことが可能な抽象度を保った高速な性能検証を、単一のプラットフォ

ーム上で行うシミュレーション装置を提供することを目的とする。

【００１１】

【課題を解決するための手段】

前記目的を達成するため本発明のシミュレーション装置は、検証処理を実行する検証実行プロセッサを備え、検証対象システムが有する検証対象プロセッサの機能をシミュレートするプロセッサモデルと、前記検証対象システムにおける前記検証対象プロセッサの周辺ハードウェアの機能をシミュレートするハードウェアモデルとを、前記検証実行プロセッサ上で動作させることにより、前記検証対象システムをシミュレートするシミュレーション装置であって、前記プロセッサモデルは、前記検証対象プロセッサの機能が、前記検証実行プロセッサで実行可能なコードで記述されたソフトウェアモデルであり、前記プロセッサモデルと前記ハードウェアモデルとの間で入出力データの受け渡しを行うインタフェースモデルを用いることを特徴とする。

【００１２】

【発明の実施の形態】

本発明は、検証対象システムにおいて、ソフトウェア実装される機能を、検証プラットフォームである検証実行プロセッサ上で動作するソフトウェアモデルによってシミュレートすることにより、ＩＳＳ（命令セットシミュレータ）を用いずにすむため、高速な性能検証が可能なシミュレーション装置を提供できる。また、このシミュレーション装置は、ソフトウェアモデルを用いることにより、検証対象プロセッサのキャッシュメモリの解析を行うことが可能な抽象度で、性能検証を行える。

【００１３】

前記インタフェースモデルは、モデル間でプロトコル変換を行うプロトコル変換部を備えたことが好ましい。抽象度の異なるモデルが混在している場合でも性能検証を高い精度で行うことが可能となるからである。なお、プロトコル変換とは、モデル間でのデータの入出力単位の大きさやタイミング等を、異なる抽象度のモデル間で調整することをいう。

【００１４】

さらに、前記インタフェースモデルは、プロセッサモデルとの間で、前記検証対象プロセッサ用の命令レベルまたはクロックレベル、基本ブロックレベル、関数レベル、またはタスクレベルのいずれかのレベルで入出力データを受け渡しすることが好ましい。これにより、様々な抽象度の異なるモデルが混在している環境でも、性能検証を行うことが可能となるからである。

【００１５】

前記プロトコル変換部は、前記モデル間のデータ転送の方向またはデータの転送単位の大きさを変換することが好ましい。これにより、異なる抽象度間のやりとりが矛盾なく行えるからである。データ転送の方向とは、データ転送がどのような命令（ｒｅａｄまたはｗｒｉｔｅ等）によるものであるかを表す。

【００１６】

前記ハードウェアモデルは、前記周辺ハードウェアの機能をＲＴレベルで記述したＲＴＬ／ＲＴＣモデル、または、前記周辺ハードウェアの機能をＲＴレベルよりも高い抽象度で記述した動作モデルであってもよい。本発明のシミュレーション装置は、インタフェースモデルを用いることにより、抽象度のレベルが異なるモデルが混在していても、高速度に性能検証が可能である。

【００１７】

また、インタフェースモデルに設けられた制御部が、前記プロセッサモデルと、動作モデルの遅延を、前記プロセッサモデルと前記ＲＴＬ／ＲＴＣモデルのプロトコルに合わせるために、前記プロセッサモデルと動作モデルの出力データの前記記憶手段における遅延を調整することが好ましい。これにより、プロセッサモデルが、動作モデルとＲＴＬ／ＲＴＣモデルが混在する環境でも性能検証のために利用可能となるからである。

【００１８】

前記ソフトウェアモデルを前記検証実行プロセッサで動作させた結果から生成されるプロ

ファイル情報を、前記検証対象プロセッサのプロファイル情報に変換することが好ましい。これにより、ソフトウェアのどのブロックがボトルネックになっているかが容易に解析できるからである。

【００１９】

また、前記の目的を達成するために、本発明にかかるシミュレーションモデル生成プログラムは、本発明にかかるシミュレーション装置用のソフトウェアモデルを生成するものであって、前記検証対象プロセッサの機能を記述したソースプログラムを、前記検証実行プロセッサで実行可能な命令セットを用いてコンパイルすることを特徴とする。このように、ソフトウェア実装される機能を、検証プラットフォームである検証実行プロセッサ上で動作するソフトウェアモデルとして生成することにより、シミュレーション装置でＩＳＳ（命令セットシミュレータ）を用いずにすみ、高速な性能検証が可能となる。

【００２０】

前記コンパイルの際に、前記ソフトウェアモデルから外部モデルへのトランザクションを、クロックサイクルレベル、命令レベル、基本ブロックレベル、関数レベル、またはタスクレベルで生成することが好ましい。これにより、生成されたソフトウェアモデルが、動作モデルとＲＴＬ／ＲＴＣモデルとが混在する環境でも性能検証を行うために利用可能となるからである。

【００２１】

前記コンパイルの際に、生成コードと前記ソースプログラムとの参照関係を抽出することが好ましい。これにより、ソースプログラムのデバッグが容易となるからである。

【００２２】

前記コンパイルの際に、前記ソフトウェアモデルを検証実行プロセッサで動作させた結果から生成されるプロファイル情報を保持するコードを挿入することが好ましい。これにより、プロファイル情報に基づいて性能最適化を行うことが可能となるからである。

【００２３】

以下、本発明の一実施形態について、図面を参照しながら説明する。

【００２４】

図１（ａ）および（ｂ）は、本実施形態にかかるシミュレーション装置の仕組みを模式的に示した図である。

【００２５】

図１（ａ）に示すように、本実施形態にかかるシミュレーション装置１０では、システムの設計初期段階において、各機能ブロックは、動作（アルゴリズム）レベルで記述された簡易な動作モデルＩ～ＩＩＩで実現されている。この動作モデルによって、各機能ブロックがどのような動作を行うか、また、システム全体としての動作が正しいか、が検証される。

【００２６】

次に、各機能ブロックが、ハードウェアおよびソフトウェアのどちらで実装されるかが決定されると、ハードウェアはＲＴＬで、ソフトウェアは検証対象システムのプロセッサで動作するオブジェクトコードにコンパイルされるＣ言語またはアセンブリ言語などで設計される。

【００２７】

ここで、例えば、動作モデルＩで実現されていた機能ブロックが、ソフトウェアで実装されることが決定されたとすると、シミュレーション装置１０は、図１（ｂ）に示すように、システム全体の検証のために従来必要であったＩＳＳを用いずに、動作モデルＩから、他の動作モデルと整合して動作するソフトウェアモデル６を生成すると共に、これに接続されるインタフェース７を生成する。これにより、ＩＳＳを用いない高速な検証環境を構築できる。

【００２８】

インタフェース７は、ソフトウェアモデル、動作モデル、およびＲＴＬ／ＲＴＣモデルの間のデータ転送の方向や大きさ等を変換し、遅延を調整する。これにより、動作モデルを

用いた検証が可能となり、高速な検証を可能にする。

【００２９】

図２に示すように、本実施形態のシミュレーション装置１０は、（１）動作モデル、（２）ソフトウェアで実装されるソフトウェアモデル、（３）ハードウェアで実装されるＲＴＬ／ＲＴＣモデル、の互いに抽象度の異なる３種類のモデルをシステム設計に用いる。

【００３０】

動作モデルは、時間概念を示す「クロック」の概念が存在しないという点で、ＲＴＬ／ＲＴＣモデルと大きく異なる。また、ソフトウェアモデルには、本来は時間概念が無い。ソフトウェアモデルは、実際のシステムではプロセッサ上で動作し、プロセッサが時間を制御するからである。しかし、本実施形態では、クロックの概念を付加したモデルをソフトウェアモデルとする。従って、動作モデル、ＲＴＬ／ＲＴＣモデル、およびソフトウェアモデルを整合させて検証するには、動作モデルへ遅延の概念を付加する必要がある。本実施形態では、システムの性能検証の時間の概念としてはクロックを最小単位とする。ところが、システム全体をＲＴＬ／ＲＴＣモデルで実現した時のシミュレーションは非常に遅い。それ以上の時間単位でブロック間のやりとりを実現することにより、シミュレーション速度の向上が図れる。

【００３１】

図２にあるように、プロセッサが制御するクロック情報を付加したソフトウェアモデルでは、命令／クロックレベルで動作するプログラムが存在するため、基本ブロックレベルまたは関数レベルでの転送を発生させることも可能である。また、意味を持った処理の単位を、本発明ではタスクとするが、タスクレベルでの転送も発生させることができる。なお、図２において、○は、そのモデルにおいて当該レベルの概念があること、△は、そのモデルにおいて当該レベルの概念があいまいであること、×はそのモデルにおいて当該レベルの概念がないこと、をそれぞれ表す。

【００３２】

一方、ハードウェアモデルであるＲＴＬ／ＲＴＣモデルは、クロックレベルでの処理しか行えない。動作モデルは、処理に必要なデータが存在すれば処理が行えるため、意味を持った処理の単位であるタスクレベルで動作する。基本ブロックレベルおよび関数レベルでは、必要なデータがそろっていることを条件として処理を行える。

【００３３】

以上より、命令／クロックレベル、基本ブロックレベル、関数レベル、タスクレベルの全てにおいて処理が行えるようにするためには、

（１）ＲＴＬ／ＲＴＣモデルへの入力は、基本ブロックレベル、関数レベル、タスクレベルのそれぞれから、命令／クロックレベルへのプロトコル変換、

（２）動作モデルへの入力は、命令／クロックレベル、基本ブロックレベル、関数レベルのそれぞれからタスクレベルへのプロトコル変換、あるいは各レベルから関数レベルまたは基本ブロックレベルへのプロトコル変換、

（３）ソフトウェアモデルへの入力は、命令／クロックレベル、タスクレベルから、基本ブロックレベルまたは関数レベルへのプロトコル変換、あるいは各レベルからタスクレベルまたは命令／クロックレベルへのプロトコル変換、

を必要に応じて行う必要がある。

【００３４】

さらには、ＲＴＬ／ＲＴＣモデルとの信号線接続のために、動作モデルの入出力信号のための信号線や信号値の変換または生成が必要になる。本発明では、入出力をトランザクションと定義する。トランザクションは、各モデルで生成され、接続されるインタフェースにより、入力対象のモデルに適合したレベルに変換され、対象のモデルに入力される。

【００３５】

このような観点から、本発明に関わるインタフェースは、必要に応じてプロトコル変換機能と信号変換機能とを実現する。

【００３６】

＜ソフトウェアモデル生成＞

本実施形態では、プロセッサＡで動作するよう設計されたプログラムを、プロセッサＢで動作するようにし、かつクロックの概念を付加したモデルを、ソフトウェアモデルとする。クロックの概念を付加するにあたり、いくつかの情報を用いる。プロセッサで動作するソフトウェアに関する遅延は、大きく分けて、データへの算術論理演算等によるプロセッサ内部での遅延と、メモリへのアクセスによるプロセッサ外部に関する遅延とに分けられる。外部へトランザクションを発生させる間隔は、プロセッサ内部での遅延に基づいており、外部に依存したデータを用いる場合には、プロセッサ外部に関する遅延後にデータが用いられる。外部に関する遅延は、ソフトウェアモデルから発生したトランザクションに対して、外部に依存して決定する。

【００３７】

以上より、ソフトウェアモデル生成では、プロセッサの情報を元に、内部での処理に関する遅延を決定し、外部へのトランザクションを、命令／クロックレベル、基本ブロックレベル、関数レベル、タスクレベルで発生させる必要がある。

【００３８】

図３に示すように、本実施形態では、発生させるトランザクションに含まれる情報は、対象となるアドレス、データの値、サイズ、方向（リードかライトか）、トランザクションの始点、終点、トランザクションの終わる時間、トランザクションが命令かデータか、といった情報の全て、あるいはこれらの情報の一部からなる組の集合とする。トランザクションには対象となるアドレス、方向など、キャッシュの解析に必要なデータが含まれるため、キャッシュの解析が可能な抽象度である。

【００３９】

ここで、プログラムからソフトウェアモデルを生成する際、トランザクションのレベルとして、クロックレベル、命令レベル、基本ブロックレベル、関数レベル、タスクレベルのそれぞれを用いたときの例を示す。

【００４０】

クロックサイクルレベルを用いた場合、プロセッサがパイプライン化されていないとき、図４（ａ）に示す代入文は、図４（ｃ）に示すトランザクションＴ１１～Ｔ１７を生成する。なお、ここで用いられる命令セットは図４（ｂ）に示すとおりである。

【００４１】

トランザクションＴ１１が対象のモデルに送られた後、（トランザクションＴ１１の発生にかかる遅延＋結果が利用できるようになる遅延）の後、トランザクションＴ１２を発生させる。トランザクションＴ１１の発生にかかる遅延は、プロセッサ内部における遅延であるので、プロセッサの持つ命令セットから決定する。例えばアドレス計算に１サイクルを費やすロード命令であった際には、トランザクションＴ１１の発生にかかる遅延は１と決定できる。トランザクションＴ１６に関する遅延は、加算演算にかかる遅延であり、同様に決定できる。

【００４２】

図６は、決定された遅延を持ったクロックサイクルレベルのトランザクションの、シミュレーションにおける発生工程を示している。シミュレーションでは図６にある機能３０を実行し、対応するトランザクションを順に、それぞれのトランザクションに関する遅延後に発生させていく。プロセッサがパイプライン化されているときには、同サイクルで発生するトランザクションを組にし、可能であれば同時に送信する。

【００４３】

本例では、図４（ａ）の式を図４（ｃ）のようにコンパイルしたが、プロセッサの持つレジスタ数や最適化の状況によっては、ロードが不要であったり、スピルコードの挿入が行われたり、異なったコンパイル結果となる。

【００４４】

また、命令レベルでトランザクションを発生させる場合、図６に示すようなトランザクションＴ１９～Ｔ２２を生成する。トランザクションＴ１９は、命令のフェッチとデータの

演算、この場合はロード処理を含んだものである。

【００４５】

図７は、命令レベルでのトランザクション発生工程を示している。シミュレーションでは、図７にある機能３１を実行し、対応するトランザクションを順に、それぞれのトランザクションに関する遅延後に発生させ、インタフェースに送る。

【００４６】

さらに、基本ブロックレベルをトランザクションのレベルとした場合、図８に示すようなトランザクションＴ２３を生成する。トランザクションＴ２３は、基本ブロック単位であるので、基本ブロックに含まれる命令列は、プログラム中で同じ回数実行される。連続する命令は連続する空間に置かれるため、一回のトランザクションでサイズを指定することにより、固まりとして表現でき、シミュレーション時に発生するトランザクションの回数を減らす効果がある。

【００４７】

図９に、基本ブロックレベルでのトランザクション発生処理工程を示す。シミュレーションでは、基本ブロックの先頭命令から始まり、基本ブロック内トランザクションデータ初期化処理（ステップＳ４０）をこの段階で行う。基本ブロックが終わるまで、基本ブロック内機能の処理（ステップＳ４２）と、基本ブロック内トランザクションデータ更新処理（ステップＳ４３）で、基本ブロック内のトランザクション情報を収集する。ステップＳ４２にて、実際のＣＰＵの処理である演算、データのロードストアを行い、機能をシミュレーションする。基本ブロックが終わるかどうかをステップＳ４１で判定し、基本ブロック内トランザクション処理（ステップＳ４４）にて、トランザクションを発生させ、インタフェースモデルに送る。

【００４８】

そして、関数レベルをトランザクションのレベルとした場合、図１０に示すようなトランザクションＴ２４を生成する。関数は基本ブロックと同等もしくはそれ以上のサイズを持つ。

【００４９】

図１１に、関数レベルでのトランザクション発生処理工程を示す。シミュレーションは、関数の先頭命令から始まる。この段階にて関数内トランザクションデータ初期化処理（ステップＳ５０）を行い、関数内で発生するトランザクションデータの初期化を行う。関数の処理が終わるまで、関数内処理機能処理（ステップＳ５１，Ｓ５３，Ｓ５５）にて関数の機能をシミュレートし、関数内トランザクションデータ更新処理（ステップＳ５２，Ｓ５４，Ｓ５６）にて関数内のトランザクション情報を収集する。関数が終わる段階、関数内トランザクションデータ処理（ステップＳ５７）にて、収集されたトランザクション情報を用いてトランザクションを発生させ、インタフェースモデルに送る。

【００５０】

最後に、タスクレベルをトランザクションのレベルとした場合、図１２に示すようなトランザクションＴ２５を生成する。タスクは意味を持った処理の固まりであるので、必要なデータが整い次第開始される。ここで述べる「意味を持った処理の固まり」とは、例えばＪＰＥＧの場合において圧縮動作のみを行う処理であり、処理単位である（８ｘ８）画素が「必要なデータ」となる。

【００５１】

図１３に、タスクレベルでのトランザクション発生処理工程を示す。まず、タスク内トランザクションデータ初期化処理（ステップＳ６０）を行う。そして、タスク内機能処理（ステップＳ６１）にて入力されたデータを用いて意味のある処理を行った後、タスクの機能実行は終了する。タスク内のトランザクション情報を蓄積し、タスク内トランザクション処理（ステップＳ６２）にて、収集したトランザクションを発生させ、インタフェースモデルに送る。

【００５２】

なお、上記各レベルにおいて、各機能処理の終了後にソフトウェアモデルからトランザク

ションを発生しているが、機能処理の途中または機能処理の前に発生する形態であっても同様の効果が得られ、本発明の技術的範囲に含まれる。

【００５３】

図１４は、ソフトウェアモデル生成の流れを示したものである。本発明では、このコード生成時に遅延情報を中間言語レベルで挿入する。ソフトウェアのプロセッサ内部または外部に依存した遅延に関するトランザクション発生のためのコードを中間言語レベルで生成・挿入し、プログラムの機能はプロセッサＢで動作させる。対比のために、図１８に、プロセッサＡ向けの従来のコード生成手順の概略を示す。プログラムから中間言語を生成し、中間言語に対して最適化し、プロセッサＡ向けのコードを生成する。

【００５４】

なお、図１４に示した例に限らず、中間言語生成の前に遅延情報の付加を行う形態とすることも可能であり、この形態も本発明の技術的範囲に含まれる。また、前記プログラム生成手順では、プログラムがＣ言語で記述されているとしたが、プログラムがアセンブリ言語で書かれている場合も同様に、遅延情報の付加が可能であり、本発明の技術的範囲に含まれる。遅延情報挿入後、プロセッサＡではなく、プラットフォームであるプロセッサＢ用にコードを生成する。挿入に関して、プロセッサのレジスタを保持する変数と、プログラムとソフトウェアモデルとの間の式の対応情報を保持し、デバッグができるようにしておく事で、性能検証だけでなく、モデルのデバッグも行える環境を構築することができる。

【００５５】

さらに各関数／タスク／変数の呼び出し回数、関数／タスク／各命令の遅延値などのプロファイル情報を保持するコードも同時に挿入することで、性能最適化に役立てることができる。

【００５６】

なお、生成された各関数／タスク／変数の呼び出し回数は、検証対象プロセッサと検証実行プロセッサで同じであるが、関数／タスク／各命令の遅延値は、検証対象プロセッサと検証実行プロセッサで異なるため、変換する必要がある。プロセッサＡ（検証対象プロセッサ）用に最適化されたコードから、各関数の命令はわかるため、命令毎の遅延値を関数毎に計算し、関数の処理時間を決定し、この時間と各関数の呼ばれた回数とを乗じたものを、関数のプロセッサＡでの遅延値とする。なお、メモリアクセスに関する遅延値は、プロセッサＢ（検証実行プロセッサ）でのシミュレーション結果から生成されたプロファイル情報を利用し、メモリアクセスに関する遅延値とする。プロセッサＡの遅延と、メモリアクセスに関する遅延との和を、その関数に関する全体の遅延とする。以上の変換で関数に関するプロファイルは行えるが、タスクに関しても同様に行える。

【００５７】

上述したソフトウェアモデルから発生するトランザクションは、すべてアドレスデータを含んでいるため、キャッシュ解析可能な抽象度を保っている。これにより、シミュレーション結果を用いて、キャッシュの解析が可能である。

【００５８】

＜プロトコル変換＞

ここで、プロトコル変換について説明する。本発明では、動作モデル、ソフトウェアモデル、ＲＴＬ／ＲＴＣモデルが接続されるため、接続を行うインタフェースモデルでは、それぞれのモデルから発生するトランザクションの変換、プロトコル変換を行う必要がある。

【００５９】

まず、ソフトウェアモデルからのトランザクションについて説明する。ソフトウェアモデルからのトランザクションの発生は、上記のソフトウェアモデル生成の説明で述べた通りである。トランザクションが発生するレベルは、クロックレベル、命令レベル、基本ブロックレベル、関数レベル、および、タスクレベルのいずれかのレベルである。このため、インターフェースモデルは、ソフトウェアモデルから発生したトランザクションを、その

転送先に適合したレベルに変換する。

【００６０】

転送先が動作モデルであるときは、動作モデルが必要とするデータ単位にトランザクションのレベルを変更する必要がある。ソフトウェアモデルから発生したトランザクションがクロックレベル、命令レベル、基本ブロックレベル、および、関数レベルのいずれかである場合は、インターフェースモデルは、それぞれのトランザクションを蓄積し、必要なデータが整い、利用可能になってから、動作モデルを駆動する必要がある。

【００６１】

図１５は、動作モデルーソフトウェアモデル間のインタフェースモデルの一実施形態について、その基本構成を示した図である。動作モデルＩＶとソフトウェアモデルＶ間に、インタフェースモデル７０が構成される。インタフェースモデル７０には、動作モデル用信号変換部７１およびソフトウェアモデル用変換部７２が備えられており、ソフトウェアモデルから動作モデルへのデータを蓄積する記憶装置７３、動作モデルからソフトウェアモデルへのデータを蓄積する記憶装置７４、ソフトウェアモデル７０における一連の動作を制御する制御装置７５から構成される。

【００６２】

信号変換部７１，７２は、データ転送の方向を合わせるために入出力を制御し、記憶装置７３，７４は、プロトコル間における遅延調整のためにデータを保持する。制御装置７５は、処理するトランザクションに含まれる遅延値に応じて、記憶装置７３，７４のデータを適当なタイミングで入出力する。例えば、ソフトウェアモデルＶからのトランザクションがクロックレベルであり、ライト（ｗｒｉｔｅ）のトランザクションである場合、信号変換部７２がライトであることを検出し、制御部７５に伝える。ライトのデータは記憶装置７３に蓄積され、動作モデルＩＶが必要とするデータサイズが整うと、記憶装置７３から制御部７５へ必要なデータが整った事を示す制御信号が伝わり、信号変換部７１を経て、動作モデルＩＶへデータが渡される。例えばＪＰＥＧの場合において動作モデルが圧縮動作のみを行うときは、処理単位である（８ｘ８）画素が、前記の「必要とするデータサイズ」である。

【００６３】

必要なデータが渡されて動作モデルが駆動され、一連の処理を終えると、出力データは信号変換部７１を経て、記憶装置７４に蓄積される。蓄積されたデータには、そのデータが置かれるべきアドレスが、制御装置７５から付加される。

【００６４】

また、ソフトウェアモデルＶからのトランザクションがリード（ｒｅａｄ）であった場合には、信号変換部７２がリードであることを検出し制御部７５に伝える。リード対象のデータは、動作モデルＩＶが駆動されて処理が終了した後には、記憶装置７４に蓄えられている。記憶装置７４に存在するデータには、制御装置７５によりそのデータが置かれるべきアドレス情報が付加されている。これにより、このアドレス情報に対応した読み出しアドレスを持つリードトランザクションに、データが与えられる。アドレス情報を持つため、記憶装置７３，７４にはＲＡＭを用いることが好ましい。ただし、ソフトウェアモデルからのトランザクションに関して、アクセスされるアドレスが常に一定の順番である場合は、記憶装置７３，７４にＦＩＦＯメモリを用いてもかまわない。これによって、アドレスに関する処理を省略することができる。

【００６５】

記憶装置７４に蓄積されるデータには、アドレスやデータと同時に、データが利用可能になるまでの遅延情報も付加される。付加された遅延情報を元に、遅延時間の経過後、信号変換部７２を経てソフトウェアモデルＶでデータをリードする。

【００６６】

図１６に、遅延タイミングチャートの一例を示す。この例は、動作モデルへのメモリアクセスに、遅延が２サイクルかかる場合である。ソフトウェアモデルから、入力データＤ１、Ｄ２、Ｄ３をそれぞれ動作モデルへ転送するライトトランザクションを発生した場合、

インタフェースモデル７０は、トランザクションごとに、記憶装置７３にデータを蓄える。この例で、動作モデルは、３つのデータが利用可能になると駆動されるものとする。３つのデータが利用可能になる時刻Ｔ１にて、動作モデルは駆動され、瞬時に結果を生成し、インタフェースモデル７０の記憶装置７４にデータを置く。ソフトウェアモデルからリードトランザクションが来ると、インタフェースモデル７０は、２サイクルの遅延を待ってから、データを出力する。

【００６７】

ソフトウェアモデルから発生したトランザクションがタスクレベルである場合は、動作モデルで必要とするデータと、ソフトウェアモデルが生成するトランザクションのデータとは同一であるため、プロトコル変換は必要でない。このため、インタフェースモデルは、遅延を付加して入出力を制御する。

【００６８】

次に、ソフトウェアモデルからのトランザクションの転送先がＲＴＬ／ＲＴＣモデルであった場合に関して説明する。ＲＴＬ／ＲＴＣモデルは、クロックの概念を持つため、ソフトウェアモデルからのトランザクションがクロックレベル以外の場合、クロックに関するプロトコル変換を実施する必要がある。また、ＲＴＬ／ＲＴＣモデルでは必要となる信号線の値生成を行う信号変換を実施する必要もある。

【００６９】

まず、クロック概念の付与であるが、連続するアドレスへのトランザクションであれば、ソフトウェアモデルからのトランザクションのデータサイズを、バスの転送可能サイズで分割することにより、１つのトランザクションを複数のトランザクションに分割する。異なるアドレスからなるトランザクションであれば、アドレスごとに分割する。例えば１つのトランザクションを第１のトランザクションおよび第２のトランザクションの２つに分割した場合、第１のトランザクションと第２のトランザクションとの間のクロックサイクルは、第１のトランザクションの終了までにかかるサイクル数とする。すなわち、１つのトランザクションが終了するまで、次のトランザクションをＲＴＬモデルへ転送しない。

【００７０】

ＲＴＬ／ＲＴＣモデルで用いられるが、より抽象度の高いレベルでは表現されない信号線（例えばブロックの選択信号線）は、トランザクションが発生したときにアクティブにして生成する。インタフェースモデルの信号変換部は、トランザクションでは複数ビットからなる単一信号を、単一ビットの複数の信号に変換したり、また単一ビットの信号を複数集めて複数ビットからなる単一信号を生成したりする。信号変換部は、ソフトウェアモデルと動作モデルとの間での信号変換処理と同様に、入出力に関する単方向－双方向変換も行う。ソフトウェアモデルから発生するトランザクションのデータには「０」か「１」の値しか含まれない。しかしＲＴＬ／ＲＴＣモデルで実現されるハードウェアブロックに対しては、シミュレーション上、不定値「ｘ」や、ハイインピーダンス状態を表す「ｚ」を入力として与える必要がある。ＲＴＬ／ＲＴＣに存在する不定値、ハイインピーダンスの概念も、インタフェースモデルの信号変換部で発生させることで、ソフトウェアモデルとＲＴＬ／ＲＴＣモデルとの間の整合をとる。具体的には、トランザクションが発生しない場合は、インタフェースモデルからは「ｘ」または「ｚ」を出力する。「ｘ」を出力するか「ｚ」を出力するかは、接続先が何かで決まり、例えば接続先がトライステート状態を持てるトライステートバスのモデルであれば「ｚ」を、それ以外は「ｘ」を出力できる。

【００７１】

動作モデルとＲＴＬ／ＲＴＣモデルとの間のインタフェースについては、前述した動作モデルーソフトウェアモデル間、ソフトウェアモデルーＲＴＬ／ＲＴＣモデル間のインタフェースを合わせて用いることにより、動作モデル―ＲＴＬ／ＲＴＣモデル間のプロトコル変換が行える。

【００７２】

尚、本実施実施形態ではインタフェースモデルにおいて記憶装置を入力用と出力用に別途用意することにより、制御を容易にしているが、入力用と出力用を同一のメモリで実装し

た形態も可能であり、本発明の技術的範囲に含まれる。

【００７３】

【発明の効果】

以上説明したとおり、本発明は、キャッシュ解析の可能な抽象度を保った性能検証を、ＩＳＳ（命令セットシミュレータ）を必要とせずに、単一の検証プラットフォームにより高速に行うことが可能なシミュレーション装置を提供することができる。

【図面の簡単な説明】

【図１】本発明のシミュレーション装置の一実施形態を示す図であり、（ａ）は機能検証工程、（ｂ）は性能検証工程をそれぞれ概念的に示した図である。

【図２】抽象度が異なる３つのモデルを示す図である。

【図３】本発明の各モデルが発生させるトランザクションに含まれる情報の図である。

【図４】プログラムからソフトウェアモデルを生成する際、トランザクションのレベルとしてクロックレベルを用いた場合に生成されるトランザクションの例を説明する図であり、（ａ）はソースプログラムの一例、（ｂ）は生成に用いられる命令セットの例、（ｃ）は生成されるトランザクションの例である。

【図５】ソフトウェアモデルにおける、クロックサイクルレベルでのトランザクション処理工程を示したものである。

【図６】プログラムからソフトウェアモデルを生成する際、トランザクションのレベルとして命令レベルを用いた場合に生成されるトランザクションの例を示す図である。

【図７】ソフトウェアモデルにおける、命令レベルでのトランザクション処理工程を示したものである。

【図８】プログラムからソフトウェアモデルを生成する際、トランザクションのレベルとして基本ブロックレベルを用いた場合に生成されるトランザクションの例を示す図である。

【図９】ソフトウェアモデルにおける、基本ブロックレベルでのトランザクション処理工程を示したものである。

【図１０】プログラムからソフトウェアモデルを生成する際、トランザクションのレベルとして関数レベルを用いた場合に生成されるトランザクションの例を示す図である。

【図１１】ソフトウェアモデルにおける、関数レベルでのトランザクション処理工程を示したものである。

【図１２】プログラムからソフトウェアモデルを生成する際、トランザクションのレベルとしてタスクレベルを用いた場合に生成されるトランザクションの例を示す図である。

【図１３】ソフトウェアモデルにおける、タスクレベルでのトランザクション処理工程を示したものである。

【図１４】ソフトウェアモデル生成の流れを示すフローチャートである。

【図１５】動作モデルーソフトウェアモデル間インタフェースモデルの構成の一例を示す図である。

【図１６】動作モデルーソフトウェアモデル間のライト、リードトランザクションでの遅延調整の一例を示す図である。

【図１７】従来のシミュレーション装置を概念的に示すものであり、（ａ）は機能検証工程、（ｂ）は性能検証工程を示す図である。

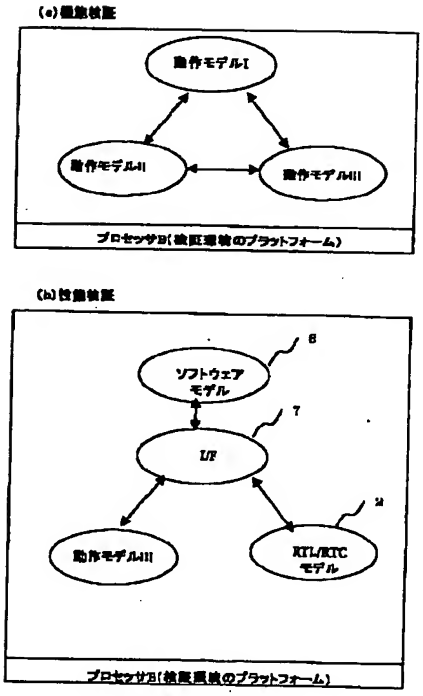
【図１８】従来のシミュレーションで、ＩＳＳで実行されるプログラムを生成する手順を示すフローチャートである。

【符号の説明】

１　プログラム
２　ＲＴＬ／ＲＴＣモデル
３　動作モデル
４　インタフェースモデル
５　ＩＳＳ
６　ソフトウェアモデル

７　インタフェースモデル
７０　インタフェースモデル
７１　信号変換部
７２　信号変換部
７３　記憶装置
７４　記憶装置
７５　制御装置

【図1】



【図2】

	タスクレベル	関数レベル	基本プロックレベル	命令/クロックレベル
動作モデル	○	△	△	×
ソフトウェアモデル	△	○	○	△
RTL/RTCモデル	×	×	×	○

【図3】

{
対象アドレス、
トランザクションの持つ値、
サイズ、
方向（リードかライトか）、
トランザクションの発生源、
転送先ID、
トランザクションの終わる時間、
トランザクションの種類（命令アクセスか、データアクセスか）
.
.
.
}

【図4】

(a)

```
a=b+c;
```

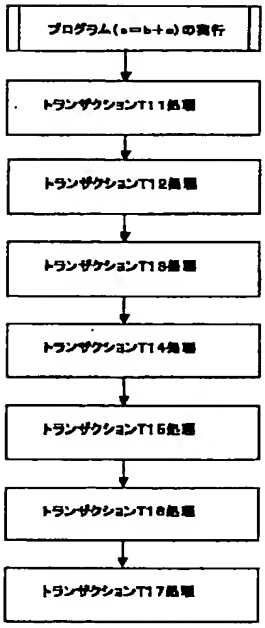
(b)

```
命令
命令Ｉ１：レジスタ１へｂの値をロード
命令Ｉ２：レジスタ２へｃの値をロード
命令Ｉ３：レジスタ１とレジスタ２の値を加算しレジスタ３に格納
命令Ｉ４：レジスタ３の値をａへストア
```

(c)

```
トランザクション（ブロックレベル）
トランザクションＴ１１：（命令Ｉ１のフェッチ）
トランザクションＴ１２：（命令Ｉ１で用いるデータのロード）
トランザクションＴ１３：（命令Ｉ３のフェッチ）
トランザクションＴ１４：（命令Ｉ２で用いるデータのロード）
トランザクションＴ１５：（命令Ｉ３のフェッチ）
トランザクションＴ１６：（命令Ｉ４のフェッチ）
トランザクションＴ１７：（命令Ｉ４で用いるデータのストア）
```

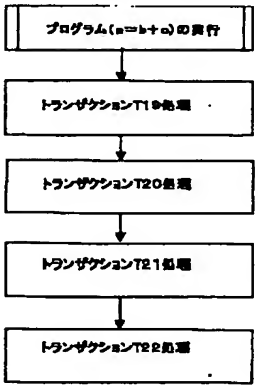
【図5】



【図6】

```
トランザクション（命令レベル）
トランザクションＴ１９：（命令Ｉ１のフェッチ、命令Ｉ１で用いるデータのロード）
トランザクションＴ２０：（命令Ｉ２のフェッチ、命令Ｉ３で用いるデータのロード）
トランザクションＴ２１：（命令Ｉ３のフェッチ）
トランザクションＴ２２：（命令Ｉ４のフェッチ、命令Ｉ４で用いるデータのストア）
```

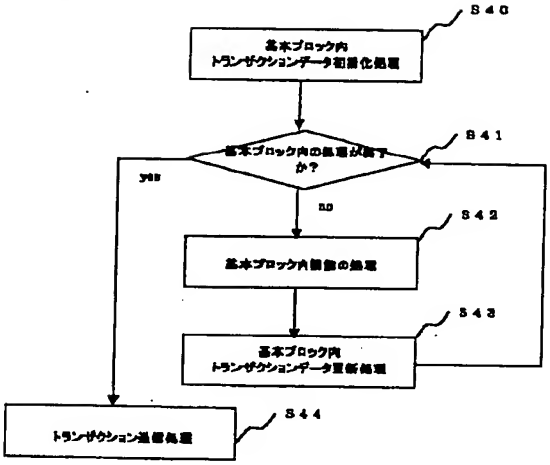
【図7】



【図8】

```
トランザクション（基本ブロックレベル）
トランザクションＴ２３：（基本ブロック先頭命令のフェッチ…、
命令Ｉ１のフェッチ、命令Ｉ２のフェッチ、命令Ｉ３のフェッチ、命令Ｉ４のフェッチ、…、基本ブロックの最終命令のフェッチ、基本ブロックの先頭命令で用いるデータのロード/ストア、…、命令Ｉ１で用いるデータのロード、命令Ｉ２で用いるデータのロード、命令Ｉ４で用いるデータのストア…基本ブロックの最終命令で用いるデータのロード/ストア）
```

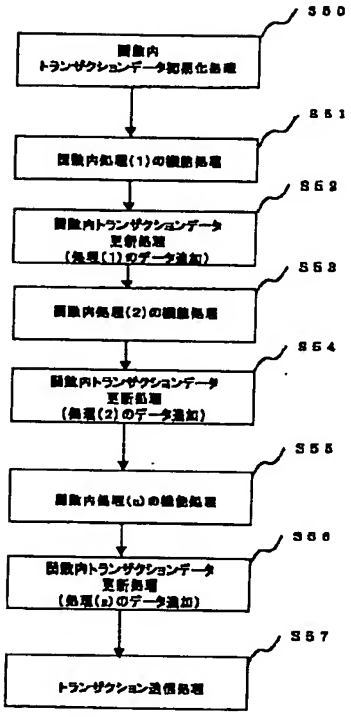
【図9】



【図10】

トランザクション（関数レベル）
トランザクションＴ２４：（関数に含まれる全命令のフェッチ、関数内に含まれる命令で使用するデータのロードストア）

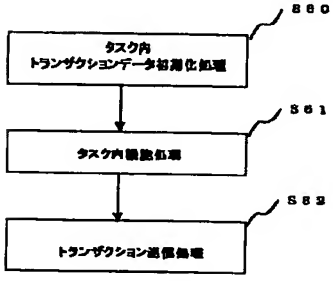
【図11】



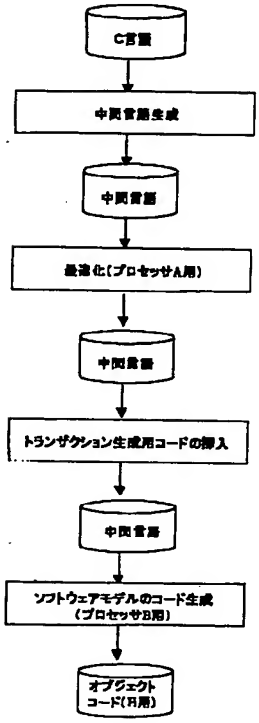
【図12】

トランザクション（タスクレベル）
トランザクションＴ２５：（タスクに含まれる全命令のフェッチ、タスク内に含まれる命令で使用するデータのロードストア）

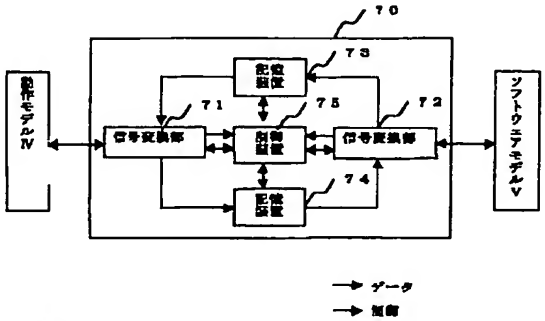
【図13】



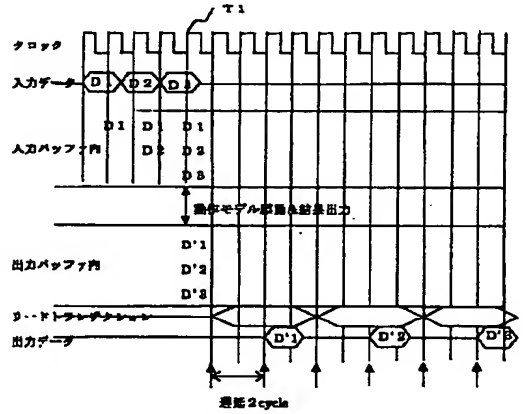
【図14】



【図15】

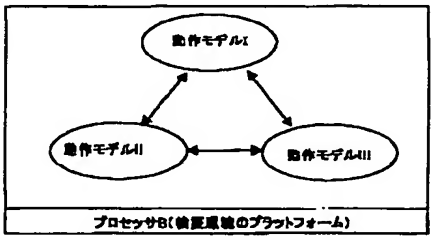


【図16】

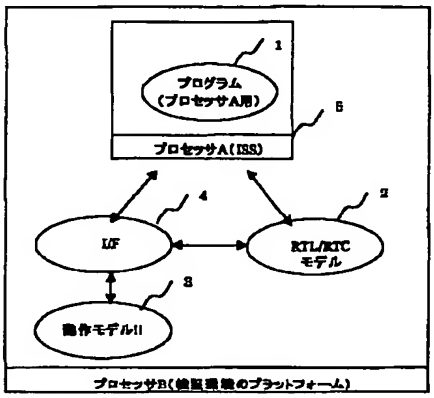


【図17】

(a)機能検証



(b)性能検証



【図18】